

Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К курсовому проектированию
по курсу «Логика и основы алгоритмизации
в инженерных задачах» на тему
«Реализация алгоритма раскрашивания графа»

13.02.24
хорошо
Гера

Выполнил:

студент группы 22ВВВ3
Гераськина Д. А.

Приняли:

Юрова О. В.
Акифьев И. В.

Пенза 2024

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

« » 20

ЗАДАНИЕ

на курсовое проектирование по курсу

Логика и основы алгоритмизации в инженерном задании
Студенту Гераховой Дарии Андреевны Группа 22ВВВ3/22ВВВ2
Тема проекта Реализация алгоритма раскраски графа

Исходные данные (технические требования) на проектирование

- Работать алгоритмов и программного обеспечения
в соответствии с требованиями задания курсового проекта.
Техническая документация должна содержать:
- 1) Постановку задачи;
 - 2) Теоретическую часть задачи;
 - 3) Описание алгоритма поставленной задачи;
 - 4) Пример ручного расчета задачи и вычислений
(на собственном участке работы алгоритма);
 - 5) Описание самой программы;
 - 6) Тест;
 - 7) Список литературы;
 - 8) Листинг программы;
 - 9) Результаты работы программы

Объем работы по курсу

1. Расчетная часть

Ручной расчёт работ алгоритма

2. Графическая часть

Схема алгоритма в формате блок-схема.

3. Экспериментальная часть

Тестирование программы;

Результаты работы программы на тестовых данных

Срок выполнения проекта по разделам

- 1 Исследование теоретической части курсового
- 2 Разработка алгоритма программы
- 3 Разработка программы
- 4 Тестирование и завершение разработки программы
- 5 Оформление пояснительной записки.
- 6
- 7
- 8

Дата выдачи задания " 6 " сентября

Дата защиты проекта " " "

Руководитель

Иванова О.В. ф.с.о.

Задание получил

" 6 " сентября 20 23 г.

Студент

Геращенко Дарья Александровна Геращенко

Содержание

Реферат	5
Введение	6
Постановка задачи.....	7
Теоретическая часть задания	8
Описание алгоритма программы	10
Описание программы	13
Тестирование	16
Ручной расчет задачи	19
Заключение	20
Список литературы	21
Приложение А.	22

Реферат

Отчет 27 стр, 6 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, РАСКРАСКА ГРАФА, ЖАДНЫЙ АЛГОРИТМ.

Цель исследования – разработка программы, реализующей алгоритм раскраски графов.

В работе рассмотрен алгоритм жадной раскраски (Greedy Coloring), который прост и эффективен, однако не гарантирует использование минимального количества цветов.

Введение

Жадный алгоритм раскраски графа – это простой и интуитивно понятный метод. Он основан на идее присвоения каждой вершине минимального возможного цвета, учитывая цвета её соседей.

Преимущества жадного алгоритма:

Простота: Жадный алгоритм легко понять и реализовать.

Быстрота: В большинстве случаев он работает быстро.

Недостатки:

Неоптимальность: Не всегда гарантирует использование минимального числа цветов. В некоторых случаях может потребоваться больше цветов, чем наилучший результат.

Чувствительность к порядку вершин: Результат может зависеть от порядка обхода вершин. Разные порядки могут привести к различным раскраскам.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2022, язык программирования – C++.

Целью данной курсовой работы является разработка программы на языке Си, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм поиска в глубину, осуществляющий поиск компоненты сильной связности орграфа.

Постановка задачи

Требуется разработать программу, которая будет раскрашивать граф, при условии, что две смежные вершины должны иметь разный цвет.

Исходный граф в программе должен задаваться матрицей смежности, причем при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должен выводиться список вершин. Необходимо предусмотреть проверку ввода, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

Задания выполняются в соответствии с вариантом №24.

Теоретическая часть задания

Граф G (рисунок 1) задается множеством вершин X_1, X_2, \dots, X_n и множеством ребер, соединяющих между собой определенные вершины. Ребра из множества A ориентированы, что показывается стрелкой, которая указывает достижимость данной вершины, граф с такими ребрами называется ориентированным графом.

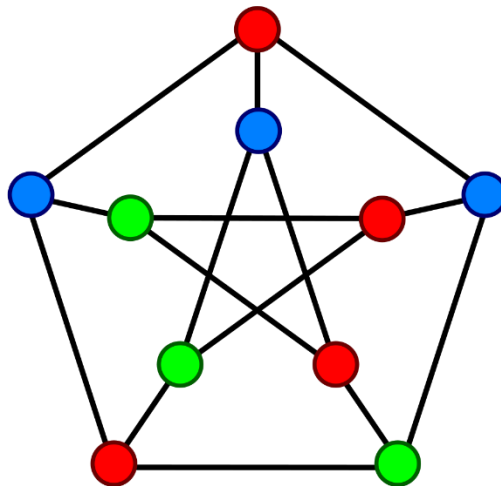


Рисунок 1 - Пример раскраски графа

При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается единицей, иначе нулем.

Существует много алгоритмов на графах, в основе которых лежит систематический перебор вершин графа, такой, что каждая вершина графа просматривается только один раз, и переход от одной вершины к другой осуществляется по ребрам графа.

Раскраска графа — теоретико-графовая конструкция, частный случай разметки графа. При раскраске элементам графа ставятся в соответствие метки с учётом определённых ограничений; эти метки традиционно называются «цветами». В простейшем случае такой способ окраски вершин графа, при котором любым двум смежным вершинам соответствуют разные цвета, называется раскраской вершин. Аналогично раскраска рёбер присваивает цвет каждому ребру так, чтобы любые два смежных ребра имели разные цвета. Наконец, раскраска областей планарного графа назначает цвет каждой области, так, что каждые две области, имеющие общую границу, не могут иметь одинаковый цвет.

Раскраска вершин — главная задача раскраски графов, все остальные задачи в этой области могут быть сведены к ней. Например, раскраска рёбер графа — это раскраска вершин его рёберного графа, а раскраска областей

планарного графа — это раскраска вершин его двойственного графа. Тем не менее, другие проблемы раскраски графов часто ставятся и решаются в изначальной постановке. Причина этого частично заключается в том, что это даёт лучшее представление о происходящем и более показательнее, а частично из-за того, что некоторые задачи таким образом решать удобнее (например, раскраска рёбер).

Раскраска графов находит применение и во многих практических областях, а не только в теоретических задачах. Помимо классических типов проблем, различные ограничения могут также быть наложены на граф, на способ присвоения цветов или на сами цвета. Этот метод, например, используется в популярной головоломке Судоку. В этой области всё ещё ведутся активные исследования.

Описание алгоритма программы

Для написания программы по раскраске графов важно иметь четкое представление о нескольких ключевых аспектах. Давайте разберемся более подробно в том, что нам необходимо для этой задачи.

1. Представление графа:

Первым шагом является выбор структуры данных для представления графа. Обычно используют матрицы смежности или списки смежности. В случае матрицы смежности, создается двумерный массив, где каждый элемент $[i][j]$ равен 1, если между вершинами i и j есть ребро, и 0 в противном случае. Списки смежности представляют собой массив списков, где каждый список содержит вершины, смежные с соответствующей вершиной.

Пример матрицы смежности:

```
int adjacencyMatrix[MAX_VERTICES][MAX_VERTICES];
```

Пример списка смежности:

```
vector<int>adjacencyList[MAX_VERTICES];
```

2. Цвета и раскраска:

Нам нужно представить цвета в программе. Для этого можно использовать целые числа, где каждое число представляет собой определенный цвет.

Пример представления цветов

```
int colors[MAX_VERTICES];
```

Также, необходимо реализовать функцию раскраски графа. Подходы к раскраске могут различаться, но для простоты рассмотрим жадный алгоритм. Он заключается в том, чтобы пройти по вершинам и присвоить каждой вершине минимальный доступный цвет, который еще не используется среди смежных вершин.

Пример жадного алгоритма раскраски на псевдокоде:

1. Проверить цвета смежных вершин
2. Найти минимальный доступный цвет среди смежных вершин
3. Присвоить вершине цвет
4. Если все цвета заняты, вернуть код ошибки.

3. Поиск доступного цвета для вершины:

Для определения минимального доступного цвета для вершины, необходимо пройти по смежным вершинам и проверить, какие цвета уже использованы.

```
int findAvailableColor(int graph[MAX_VERTICES][MAX_VERTICES], int vertices, int vertex) {  
    // Массив, отмечающий использованные цвета  
    bool usedColors[MAX_COLORS] = {false};  
    // Проверить цвета смежных вершин  
    for (int adjacentVertex = 0; adjacentVertex < vertices;  
        ++adjacentVertex) {  
        if (graph[vertex][adjacentVertex] != -1) {  
            usedColors[adjacentVertex] = true;  
        }  
    }  
    for (int color = 0; color < MAX_COLORS; ++color) {  
        if (!usedColors[color]) {  
            return color;  
        }  
    }  
    return -1;  
}
```

```

        usedColors[colors[adjacentVertex]] = true;
    }
}
// Найти минимальный доступный цвет
for (int color = 0; color < MAX_COLORS; ++color) {
    if (!usedColors[color]) {
        return color;
    }
}
// Если все цвета заняты, вернуть -1 или другое значение,
указывающее на ошибку
return -1;
}

```

Ввод и вывод:

Добавим функции для ввода данных о графе и вывода результатов раскраски.

```

// Ввод графа
void inputGraph(int graph[MAX_VERTICES][MAX_VERTICES], int
vertices) {
    // Реализация ввода графа
}
// Вывод результатов раскраски
void outputColoring(int vertices) {
    for (int vertex = 0; vertex < vertices; ++vertex) {
        cout << "Vertex " << vertex << " is colored with
color " << colors[vertex] << endl;
    }
}

```

Жадный алгоритм раскраски на псевдокоде:

Решение задачи о раскраске графа.

Переборный алгоритм

Вход: $G=(V, A)$ - неориентированный граф, представленный матрицей смежности.

Выход: $num[v]$ - вектор цветов вершин.

Алгоритм ПРГ:

1. Создать список вершин $Q=V$, упорядоченных по убыванию степеней;
2. Для всех v $num[v]=0$;
3. Выбрать первый цвет $r=1$;
4. Выбрать первую v из Q ;
5. Окрасить вершину $num[v]=r$;
6. ПОКА не окрашены все вершины;
 7. Выбрать из Q элемент u ;
 8. ЕСЛИ u НЕ СМЕЖНА с окрашенными в использованные цвета $r=1\dots$;
 9. ТО
 10. {

```
11. окрашиваем в num[u] = min (из использованных r);
12. Q -= u;
13. }
14. ИНАЧЕ
15. {
16. r++;
17. }
```

Описание программы

Для написания данной программы использован язык программирования C++. Язык программирования C++ - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

Проект был создан в виде консольного приложения Win32 (Visual C++).

Данная программа является многомодульной, поскольку состоит из нескольких функций: `main()`, `AddFromKeyBoard()` и класса `Graph` с методами: `addEdge()`, `printGraph()`, `printGraphToFile()`, `colorGraph()`, `dfsColoring()`, `printColors()`, `printColorsToFile()`.

Работа программы начинается с запроса количества вершин графа. Далее пользователю предлагается на выбор автозаполнение ребер или ручная (с клавиатуры). Если пользователь выбирает автоматический режим, программа создает циклический граф. После генерации, созданный граф выводится на экран.

```
SetConsoleCP(1251);
SetConsoleOutputCP(1251);

int numVertices;
sf.safe_input("Введите количество вершин графа: ",
numVertices);
if (numVertices < 1)
{
    std::cout << "Вы задали число вершин 0. Граф пустой,
его невозможно раскрасить. Программа завершена" << std::endl;
    return 0;
}

Graph g(numVertices);

bool automatical = sf.end_request("Заполнить граф
автоматически? (да|нет)\n > ");
//Автоматическое заполнение - инициализирует циклический
граф
if (automatical)
{
    for (int i = 0; i < numVertices-1; ++i)
    {
        g.addEdge(i, i + 1);
    }
}
```

```

        g.addEdge(numVertices - 1, 0);
    }
    //Ручное заполнение - берет данные с клавиатуры
    else
    {
        AddFromKeyBoard(g);
    }

    std::cout << "Граф:" << std::endl;
    g.printGraph();

```

После чего программа начинает раскрашивать граф, пока не обойдет все вершины с помощью функции dfsColoring().

```
g.colorGraph();
```

По окончании раскраски графа жадным алгоритмом пользователю предлагается сохранить результат раскраски.

```

bool save = sf.end_request("Сохранить результат? (да|нет)\n >
");

    if (save)
    {
        g.printGraphToFile();
        g.printColorsToFile();
    }

```

Стоит обратить внимание, что в данном проекте была использована сторонняя библиотека `safe_input.h`, которая умеет проверять ввод пользователя на «дурака» и сама запрашивает повторный ввод в случае ввода некорректных данных. Так же она позволяет обрабатывать простые ответы пользователя «да, нет, 1, 0» на запрошенный вопрос.

Ниже можно увидеть оформление начального запроса и дальнейшие действия с ним.

```

D:\AdditionalFolderC++\Project\РаскраскаГрафа_ДашаГераськина\Debug\РаскраскаГрафа_ДашаГераськина.exe
Введите количество вершин графа: 5
Заполнить граф автоматически? (да|нет)
> да
Граф:
Вершина 0: 1 4
Вершина 1: 0 2
Вершина 2: 1 3
Вершина 3: 2 4
Вершина 4: 0 3
Хроматическое число: 1
Вершина 0 раскрашена цветом 0
Вершина 1 раскрашена цветом 1
Вершина 2 раскрашена цветом 0
Вершина 3 раскрашена цветом 1
Вершина 4 раскрашена цветом 2
Сохранить результат? (да|нет)
> _

```

Рисунок 2 - Автоматическое заполнение графа


```
Введите количество вершин графа: 6
Заполнить граф автоматически? (да|нет)
> нет
Добавление ребра.
Введите начало ребра(откуда): 0
Добавление ребра.
Введите конец ребра(куда): 1
Ребро успешно добавлено!
Продолжить? (да|нет)
> да
Добавление ребра.
Введите начало ребра(откуда): 1
Добавление ребра.
Введите конец ребра(куда): 0
Ребро успешно добавлено!
Продолжить? (да|нет)
> да
Добавление ребра.
Введите начало ребра(откуда): 2
Добавление ребра.
Введите конец ребра(куда): 3
Ребро успешно добавлено!
Продолжить? (да|нет)
> да
Добавление ребра.
Введите начало ребра(откуда): 3
Добавление ребра.
Введите конец ребра(куда): 2
Ребро успешно добавлено!
Продолжить? (да|нет)
> да
Добавление ребра.
Введите начало ребра(откуда): 4
Добавление ребра.
Введите конец ребра(куда): 5
Ребро успешно добавлено!
Продолжить? (да|нет)
> да
Добавление ребра.
Введите начало ребра(откуда): 5
Добавление ребра.
Введите конец ребра(куда): 4
Ребро успешно добавлено!
Продолжить? (да|нет)
> нет

Граф:
Вершина 0: 1
```

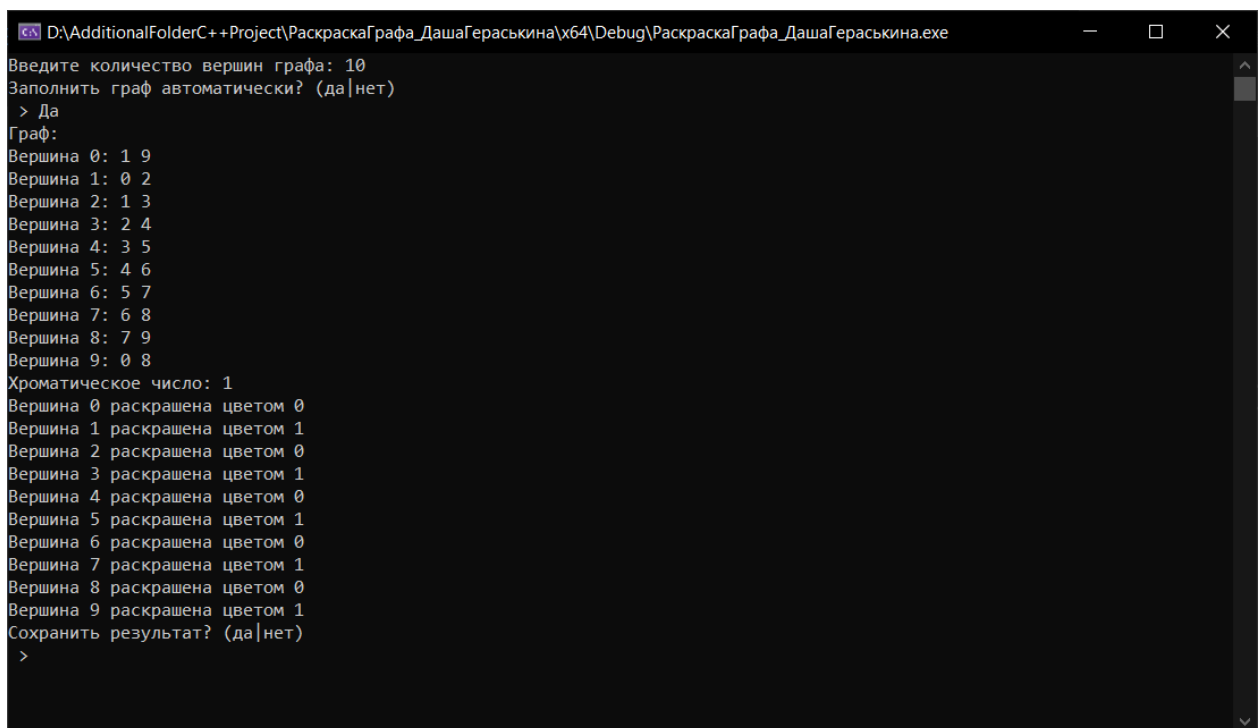
Рисунок 3 - Ручное заполнение ребер

Тестирование

Среда разработки Microsoft Visual Studio 2023 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и вывод разных вариантов раскрасок графа.



```
D:\AdditionalFolderC++\Project\РаскраскаГрафа_ДашаГераськина\x64\Debug\РаскраскаГрафа_ДашаГераськина.exe
Введите количество вершин графа: 10
Заполнить граф автоматически? (да|нет)
> Да
Граф:
Вершина 0: 1 9
Вершина 1: 0 2
Вершина 2: 1 3
Вершина 3: 2 4
Вершина 4: 3 5
Вершина 5: 4 6
Вершина 6: 5 7
Вершина 7: 6 8
Вершина 8: 7 9
Вершина 9: 0 8
Хроматическое число: 1
Вершина 0 раскрашена цветом 0
Вершина 1 раскрашена цветом 1
Вершина 2 раскрашена цветом 0
Вершина 3 раскрашена цветом 1
Вершина 4 раскрашена цветом 0
Вершина 5 раскрашена цветом 1
Вершина 6 раскрашена цветом 0
Вершина 7 раскрашена цветом 1
Вершина 8 раскрашена цветом 0
Вершина 9 раскрашена цветом 1
Сохранить результат? (да|нет)
>
```

Рисунок 4 - Тестирование автоматического заполнения

```
Консоль отладки Microsoft Visual Studio
Введите количество вершин графа: 9
Заполнить граф автоматически? (да|нет)
> да
Граф:
Вершина 0: 1 8
Вершина 1: 0 2
Вершина 2: 1 3
Вершина 3: 2 4
Вершина 4: 3 5
Вершина 5: 4 6
Вершина 6: 5 7
Вершина 7: 6 8
Вершина 8: 0 7
Хроматическое число: 1
Вершина 0 раскрашена цветом 0
Вершина 1 раскрашена цветом 1
Вершина 2 раскрашена цветом 0
Вершина 3 раскрашена цветом 1
Вершина 4 раскрашена цветом 0
Вершина 5 раскрашена цветом 1
Вершина 6 раскрашена цветом 0
Вершина 7 раскрашена цветом 1
Вершина 8 раскрашена цветом 2
Сохранить результат? (да|нет)
> нет

D:\AdditionalFolderC++Project\РаскраскаГрафа_ДашаГераськина\x64\Debug\РаскраскаГрафа_ДашаГераськина.exe (процесс 3656) з
авершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно...
```

Рисунок 5 - Тестирование нечетного автоматического заполнения

```
D:\AdditionalFolderC++Project\РаскраскаГрафа_ДашаГераськина\x64\Debug\РаскраскаГрафа_ДашаГераськина.exe
Введите количество вершин графа: 3
Заполнить граф автоматически? (да|нет)
> 1
Граф:
Вершина 0: 1 2
Вершина 1: 0 2
Вершина 2: 0 1
Хроматическое число: 1
Вершина 0 раскрашена цветом 0
Вершина 1 раскрашена цветом 1
Вершина 2 раскрашена цветом 2
Сохранить результат? (да|нет)
> .
```

Рисунок 6 - Тестирование элементарного графа

Таблица 1 – Описание поведения программы при тестировании

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Получение количества вершин графа	Верно
Выбор способа генерации	Вывод сообщения о выборе способа заполнения ребер графа	Верно
Генерация графа и его раскраска	Вывод раскрашенного графа	Верно
Запрос на сохранение результата	Сохранение результата	Верно

В результате тестирования было выявлено, что программа успешно проверяет данные на соответствие необходимым требованиям.

Ручной расчет задачи

Проведем проверку программы посредством ручных вычислений на примере графа с 3, 9 и 10 вершинами (рисунки 4, 5, 6).

В циклическом графе, состоящем из 3 вершин, все вершины являются соседними друг другу, следовательно, хроматическое число будет равно 3 и граф будет раскрашен в 3 разных цвета. На рисунке 6 прекрасно видно, что программа раскрасила граф в 3 разных цвета.

В циклическом графе, состоящем из 9 вершин, что является нечетным количеством, следовательно последняя вершина (или любая одна вершина) будет иметь соседа с тем же цветом, следовательно хроматическое число такого графа будет равно 3. На рисунке 5 видно, что программа успешно раскрасила наш граф в 3 цвета так, что начиная с первого по предпоследний элемент чередуются цветами 0 и 1, а последняя вершина имеет цвет 2.

В циклическом графе, состоящем из 10 вершин, все вершины можно разделить на пары, таким образом хроматическое число такого графа будет равняться 2. На рисунке 4 отчетливо видно, что программа успешно раскрасила поочередно вершины графа в цвета 0 и 1.

Таким образом можно сделать вывод, что программа работает исправно и блестяще исполняет свою задачу.

Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая жадный алгоритм раскраски графа в Microsoft Visual Studio 2023.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания графов на основе векторов. Приобретены навыки по осуществлению алгоритма поиска в глубину. Углублены знания программирования на языке C++.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

Список литературы

1. Стэнли Липпман. С++ Primer Эддисон-Уэсли, 2017. - 976 с.
2. Татт У. Теория графов. М.: Мир, 1988, 424 стр
3. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.
4. Харви Дейтел, Пол Дейтел. Как программировать на С/С++. 2009 г.
5. Г. Хаггарти «Дискретная математика для программистов» - Издание 2-е, 2001 г. – 401 с.

Приложение А. Листинг программы.

Main.cpp

```
#include <iostream>
#include <vector>
#include <set>
#include <string>
#include <windows.h>
#include <fstream>
#include "safe_input.h"

SafeInput sf;

class Graph
{
public:
    Graph(int vertices) : V(vertices), chromaticNumber(0)
    {
        adjacencyMatrix.resize(V, std::vector<int>(V, 0)); //
        Матрица смежности, заполненная 0
    }

    // Функция добавления ребра
    bool addEdge(int v, int w)
    {
        // Проверка на границы
        if (v < 0 || v >= V || w < 0 || w >= V) return 1;
        // Устанавливаем ребра
        adjacencyMatrix[v][w] = 1;
        adjacencyMatrix[w][v] = 1;
        return 0;
    }

    // Функция вывода графа в консоли
    void printGraph()
    {
        for (int i = 0; i < V; ++i)
        {
            // Вывод текущей вершины
            std::cout << "Вершина " << i << ": ";
            for (int j = 0; j < V; ++j)
            {
                // Если вершины смежны
                if (adjacencyMatrix[i][j] == 1)
                {
                    std::cout << j << " ";
                }
            }
            std::cout << std::endl;
        }
    }
}
```

```

// Функция сохранения результатов в файле
void printGraphToFile()
{
    // Ввод имени файла
    std::string fileName;
    std::cout << std::endl << "Введите имя файла для
сохранения графа: ";
    std::cin >> fileName;
    std::ofstream file(fileName);
    for (int i = 0; i < V; ++i)
    {
        // Ввод связей в файл
        file << "Вершина " << i << ": "; // Вывод текущей
вершины
        for (int j = 0; j < V; ++j)
        {
            // Если вершины смежны
            if (adjacencyMatrix[i][j] == 1)
            {
                file << j << " ";
            }
        }
        file << std::endl;
    }
    file.close();
}

// Функция раскраски графа и получения хроматического числа
void colorGraph()
{
    colors.resize(V, -1); // Помечаем все вершины
нераскрашенными
    chromaticNumber = 0; // Хроматическое число

    for (int i = 0; i < V; ++i)
    {
        // Если вершина не раскрашена
        if (colors[i] == -1)
        {
            dfsColoring(i); // Вызов функции обхода и поиска
цвета
            ++chromaticNumber; // Увеличиваем хроматическое
число
        }
    }

    std::cout << std::endl << "Хроматическое число: " <<
chromaticNumber << std::endl;
    printColors();
}

```

```

private:
    int V; // Вершины
    int chromaticNumber; // Для поиска нового цвета
    std::vector<std::vector<int>> adjacencyMatrix; // Матрица
    std::vector<int> colors; // Уникальные цвета

    // Поиск в глубину
    void dfsColoring(int vertex)
    {
        static std::set<int> availableColors; // Множество цветов

        for (int i = 0; i < V; ++i)
        {
            // Если вершина смежна и нераскрашена
            if (adjacencyMatrix[vertex][i] == 1 && colors[i] != -
1)
            {
                availableColors.insert(colors[i]); // Добавляем в
множество
            }

            int chosenColor = 0;

            // Находим новый цвет для вершины
            while (availableColors.count(chosenColor))
            {
                ++chosenColor;
            }

            colors[vertex] = chosenColor; // Сохранение цвета
            availableColors.clear(); // Очищение множества

            // Рекурсивный вызов для всех смежных вершин
            for (int i = 0; i < V; ++i)
            {
                if (adjacencyMatrix[vertex][i] == 1 && colors[i] == -
1)
                {
                    dfsColoring(i);
                }
            }
        }

    public:
        // Функция вывода цветов в консоль
        void printColors()
        {
            for (int i = 0; i < V; ++i)
            {

```

```

        std::cout << "Вершина " << i << " раскрашена цветом "
<< colors[i] << std::endl; // Вывод цвет раскраски соответствующей
вершины
    }
}
// Функция ввода цветов в файл
void printColorsToFile()
{
    // Ввод имени файла
    std::cout << std::endl << "Введите имя файла для
сохранения цветов графа: ";
    std::string fileName;
    std::cin >> fileName;
    std::ofstream file(fileName);
    for (int i = 0; i < V; ++i)
    {
        file << "Вершина " << i << " раскрашена цветом " <<
colors[i] << std::endl; // Ввод цвет раскраски соответствующей
вершины
    }
    file.close();
}
};

// Функция добавления ребра между вершинами
void AddFromKeyBoard(Graph& g)
{
    bool resume = 1; // Для продолжения заполнения, изначально в
режиме добавления
    while (resume) // Пока добавляем
    {
        int input;
        int output;
        sf.safe_input("Добавление ребра.\nВведите начало
ребра(откуда): ", input); // Начальная вершина для ребра
        sf.safe_input("Добавление ребра.\nВведите конец
ребра(куда): ", output); // Конечная вершина для ребра

        // Если добавление не удалось
        if (g.addEdge(input, output))
        {
            std::cout << "Введены некорректные данные, попробуйте
снова." << std::endl;
            //resume = true;
            //continue;
        }
        else // Иначе
        {
            std::cout << "Ребро успешно добавлено!" << std::endl;
        }
        resume = sf.end_request("Продолжить? (да|нет)\n > "); //
Проверка на продолжение
    }
}

```

```

    }
    std::cout << std::endl << std::endl;
}

// Основная функция
int main()
{
    // Ввод и вывод на русском
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    // Вывод титульника
    std::cout << "Пензенский государственный университет" <<
std::endl;
    std::cout << "\"Кафедра \"Вычислительная техника\"" <<
std::endl;
    std::cout << "Курсовая работа" << std::endl;
    std::cout << "По курсу \"Логика и основы алгоритмизации в
инженерных задачах\"" << std::endl;
    std::cout << "На тему \"Реализация алгоритма раскрашивания
графа\"" << std::endl;
    std::cout << "Приняли: Юрова О.В. и Акифьев И.В." <<
std::endl;
    std::cout << "Выполнила: Гераськина Дарья Андреевна, учебная
группа 22ВВВ3(22ВВП2)" << std::endl << std::endl;
    //

    int numVertices;
    sf.safe_input("Введите количество вершин графа: ",
numVertices); // Ввод кол-ва вершин
    while (numVertices < 1) // Если вершин меньше 1, входим в цикл
    {
        std::cout << std::endl << "Вы задали число вершин меньше
1. Граф пустой, его невозможно раскрасить. Повторите попытку." <<
std::endl << std::endl;
        sf.safe_input("Введите количество вершин графа: ",
numVertices); // Ввод кол-ва вершин
    }

    Graph g(numVertices); // Создаем граф в соответствии с
введенным числом вершинам

    bool automatical = sf.end_request("\nЗаполнить граф
автоматически? (да|нет)\n > "); // Как будет происходить
заполнение графа

    std::cout << std::endl;

    // Автоматическое заполнение - инициализируем циклический граф
    if (automatical)
    {
        for (int i = 0; i < numVertices-1; ++i)

```



```

        {
            g.addEdge(i, i + 1); // Соединяем текущую и следующую
вершину
        }
        g.addEdge(numVertices - 1, 0); // Соединение первой и
последней вершины, для заикливания
    }
    else // Иначе ручное заполнение - берем данные с клавиатуры
    {
        AddFromKeyBoard(g);
    }

    std::cout << "Граф:" << std::endl;

    g.printGraph(); // Выводим связи между вершинами

    g.colorGraph(); // Выводим цвет раскраски

    std::cout << std::endl;

    bool save = sf.end_request("Сохранить результат? (да|нет)\n >
"); // Хочет ли пользователь сохранить результат в файле

    // Если пользователь выбрал "да"
    if (save)
    {
        g.printGraphToFile(); // Выводим связи между вершинами
        g.printColorsToFile(); // Выводим цвет раскраски
    }

    return 0;
}

```